

FIG. 1

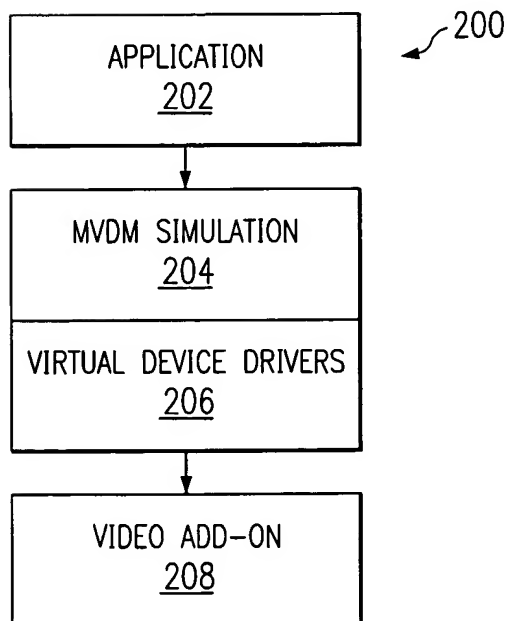


FIG. 2

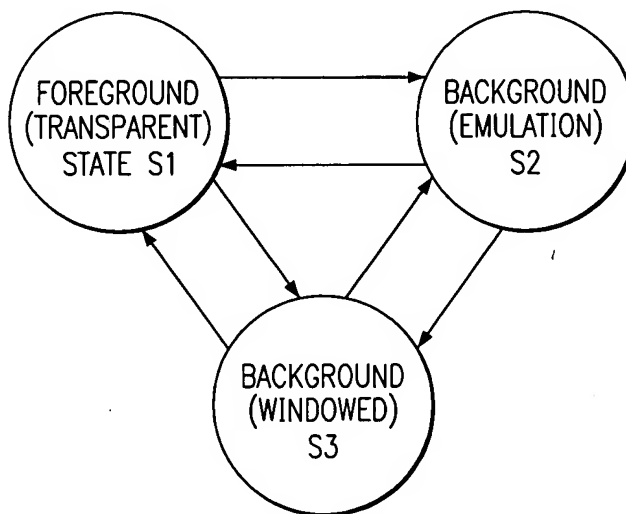


FIG. 3

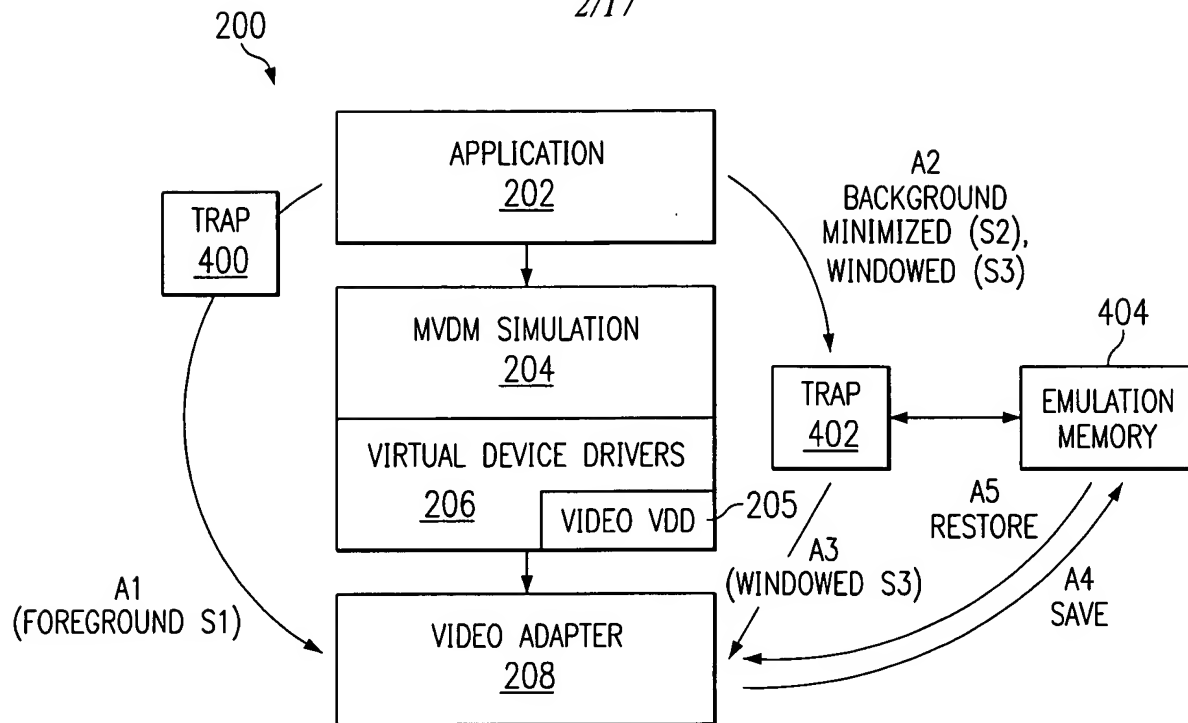


FIG. 4

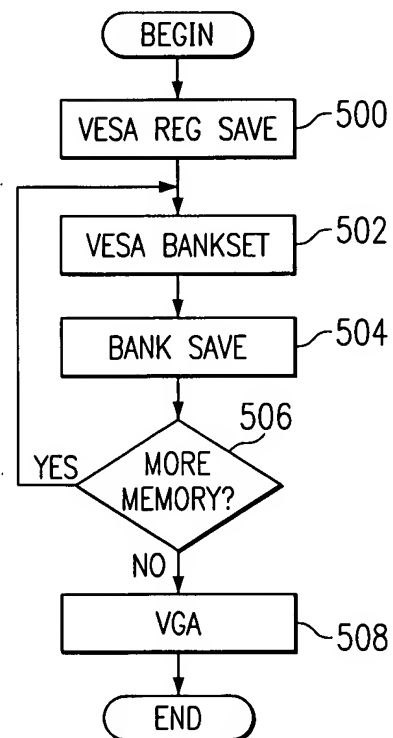


FIG. 5

3/17

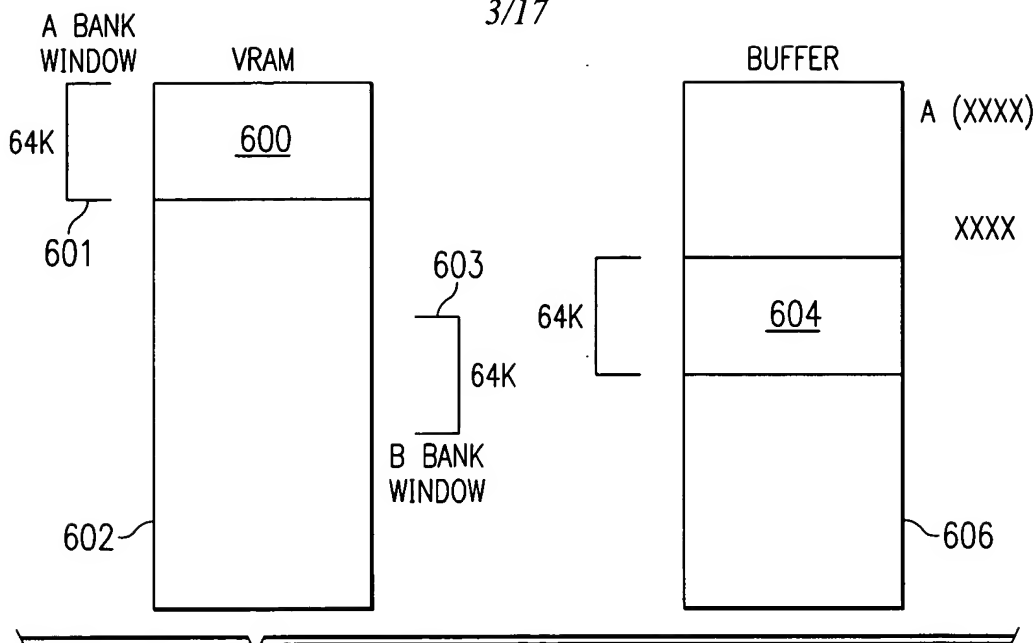


FIG. 6

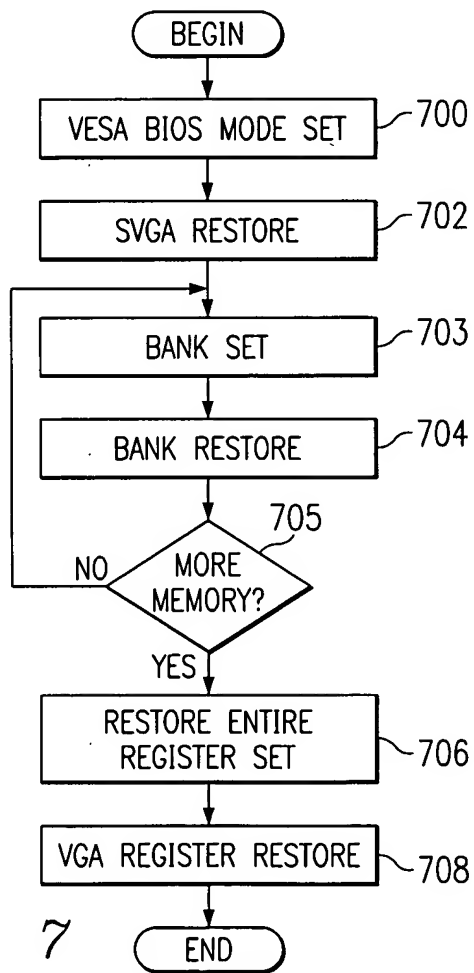


FIG. 7

*FIG. 8A*

4/17

```

/*****
 *
 * FUNCTION NAME = vvUserFgndSetMode
 *
 * DESCRIPTION
 *   Save client machine CPU register state
 *   Save video BIOS data area
 *   Setup a VGA (or possibly VESA) BIOS call to set the current
 *   client video mode in order to restore the VDM's state.
 *
 *****/

vvUserFgndSetMode()
{
    /* New art */
    Save client CPU register state
    Save video BIOS data area
    setup VGA (or possibly VESA) BIOS call to set the current
    client video mode
    return to
    vvUserFgndLogicalLineLength
}

/*****
 *
 * FUNCTION NAME = vvUserFgndLogicalLineLength
 *
 * DESCRIPTION
 *   Setup a VESA BIOS call to set the logical scan line length
 *   Useful for VESA BIOS not implementing full register restore.
 *
 *****/

vvUserFgndLogicalLineLength()
{
    /* New art */
    inject vesa call to restore
    logical scan length start registers from saved area
    return to
    vvUserFgndDisplayStart,
}

/*****
 *
 * FUNCTION NAME = vvUserFgndDisplayStart
 *
 * DESCRIPTION
 *   Setup a VESA BIOS call to set the display start registers
 *   inserts the int 10 instruction, and arms a return to
 *   vvUserFgndBankCopy.
 *
 *****/

vvUserFgndDisplayStart()
{
    /* New art */
    inject vesa call to restore display start registers from saved area
    return to
    vvUserFgndRegsSet,
}

```

800

802

804

*FIG. 8B*

5/17

```

/*****
 *
 * FUNCTION NAME = vvUserFgndRegsSet
 *
 * DESCRIPTION
 *   Setup a VESA BIOS call to restore the clients adapter registers
 *
 *****/
vvUserFgndRegsSet()
{
    /* New art */
    inject vesa call to restore client adapter registers from saved area
    return to
        vvUserFgndBankSet1st,
}

/*****
 *
 * FUNCTION NAME = vvUserFgndBankSet1st
 *
 * DESCRIPTION
 *   Setup a VESA BIOS call to set the VRAM bank number to 0;
 *
 *****/
vvUserFgndBankSet1st()
{
    /* New art */
    if( Mode uses Linear Frame Buffer )
        transfer LINEAR buffer contents to VRAM from saved area
        inject vesa call to set A Bank to saved A bank
        return to
            vvUserFgndBankBSet,
    else
        pvd->VdmUser.IBankCopyNextBank = 0;
        inject vesa call to set A Bank to next bank # for restore
        return to
            vvUserFgndBankCopySetBBank,
}

/*****
 *
 * FUNCTION NAME = vvUserFgndBankCopySetBBank
 *
 * DESCRIPTION
 *   Set the B Bank Window if it is needed for read/write operations.
 *   Most adapters only have an A Bank.
 *   A few have an A Bank for reading and a B Bank for writing,
 *   or vice versa.
 *
 *****/
vvUserFgndBankCopySetBBank()
{
    /* New art */
    inject vesa call to set B Bank to next bank # for restore
    return to
        vvUserFgndBankCopy,
}

```

806

808

810

FIG. 8C

6/17

```

/*****
 *
 * FUNCTION NAME = wvUserFgndBankCopy
 *
 * DESCRIPTION
 *   Transfers virtual memory to the VRAM bank,
 *   and then setup a VESA BIOS call to access the next A bank.
 *
 *   On the last pass, it does the transfer of virtual memory to the VRAM
 *   bank, and then setup a VESA BIOS call to set the bank
 *   number to the clients current A bank number.
 *****/
wvUserFgndBankCopy()
{
    /* Prior art */
    transfer one (current) bank of VRAM from saved area
    /* New art */
    increment bank number
    if( copy bank < total banks )
        inject vesa call to set A Bank to next bank # for restore
        return to
        wvUserFgndBankCopySetBBank,
    else
        inject vesa call to set A Bank to client bank #
        return to
        wvUserFgndBankBSet,
}

/*****
 *
 * FUNCTION NAME = wvUserFgndBankBSet
 *
 * DESCRIPTION
 *   Setup a VESA BIOS call to set the B bank
 *   number to the clients current bank number.
 *   Most adapters only have an A Bank.
 *   A few have an A Bank for reading and a B Bank for writing,
 *   or vice versa.
 *   Useful for VESA BIOS not implementing full register restore.
 *****/
wvUserFgndBankBSet()
{
    inject vesa call to set B Bank to saved bank #
    return to
        wvUserFgndRegsSetAtEnd,
}

/*****
 *
 * FUNCTION NAME = wvUserFgndRegsSetAtEnd
 *
 * DESCRIPTION
 *   Setup a VESA BIOS call to restore the client adapter
 *   register set to clean up the registers changed
 *   during the restoring the VRAM banks.
 *****/
wvUserFgndRegsSetAtEnd()
{
    inject vesa call to restore registers from saved state
    return to FgndFinish
}

```

812

814

816

*FIG. 8D*

```

/*****
 *
 * FUNCTION NAME = vvUserFgndFinish
 *
 * DESCRIPTION
 *   Finish foreground switch in VDM's context.
 *   Restore the VGA register state directly.
 *   Useful for VESA BIOS not implementing full register restore.
 *   Restore client machine CPU register state saved
 *   Restore video BIOS data area saved
 *   Switch trapping behavior to transparent real hardware access.
 *
 *****/
vvUserFgndFinish()
{
    /* Prior art */
    restore VGA register state
    /* New art */
    restore client machine CPU register state saved
    restore video BIOS data area saved
    /* Prior art */
    switch trapping behavior to transparent real hardware access
    thaw VDM when in unemulatable (SVGA) video mode.
}

```

818

*FIG. 9A*

```

/*****
 *
 * FUNCTION NAME = vvUserBgndSaveSizeQuery
 *
 * DESCRIPTION
 *   Save the VGA register state directly.
 *   Useful for VESA BIOS not implementing full register save.
 *   Save client machine CPU register state
 *   Save video BIOS data area
 *   Setup a VESA BIOS call to get the clients SVGA regs save area size.
 *
 *****/
vvUserBgndSaveSizeQuery()
{
    /* New art */
    Save client machine CPU register state
    Save video BIOS data area
    inject VESA BIOS all to get client SVGA regs save area size
    return to
        vvUserBgndRegsGet,
}

/*****
 *
 * FUNCTION NAME = vvUserBgndRegsGet
 *
 * DESCRIPTION
 *   Checks the SVGA regs save area size returned.
 *   If the DOS allocated save area is large enough,
 *   then it issues the VESA BIOS call to save the SVGA registers.
 *   Setup a VESA BIOS call to save adapter register state.
 *
 *****/
vvUserBgndRegsGet()
{
    /* New art */
    Setup a VESA BIOS call to save adapter register state.
    return to
        vvUserBgndLogicalLineLength,
}

```

900

902



*FIG. 9B*

```

/*****
 *
 * FUNCTION NAME = vvUserBgndLogicalLineLength
 *
 * DESCRIPTION
 *     Setup a VESA BIOS call to get the clients VRAM bank number.
 *
 *****/
vvUserBgndLogicalLineLength()
{
    /* New art */
    Setup a VESA BIOS call to get the clients VRAM bank number.
    return to
        vvUserBgndDisplayStart,
}
/*****
 *
 * FUNCTION NAME = vvUserBgndDisplayStart
 *
 * DESCRIPTION
 *     Save returned logical line length values.
 *     Setup a VESA BIOS call to get the clients display start offset.
 *
 *****/
vvUserBgndDisplayStart()
{
    /* New art */
    Save returned logical line length values.
    Setup a VESA BIOS call to get the clients display start offset.
    return to
        vvUserBgndBankGet,
}
/*****
 *
 * FUNCTION NAME = vvUserBgndBankGet
 *
 * DESCRIPTION
 *     Save returned display start values.
 *     Setup a VESA BIOS call to get the clients VRAM A bank number.
 *
 *****/
vvUserBgndBankGet()
{
    /* New art */
    Save returned display start values.
    Setup a VESA BIOS call to get the clients VRAM A bank number.
    return to
        vvUserBgndBankBGet,
}

```

904

906

908

FIG. 9C

10/17

```

/*****
 *
 * FUNCTION NAME = vvUserBgndBankBGet
 *
 * DESCRIPTION
 *     Save returned A bank number.
 *     Setup a VESA BIOS call to get the clients VRAM B bank number.
 *
 *****/
vvUserBgndBankBGet()
{
    /* New art */
    Save returned A bank number.
    set current copy bank to -1
    Setup a VESA BIOS call to get the clients VRAM B bank number.
    return to
        vvUserBgndBankCopy
}
/*@V4.0JAN01*/
/*****
 *
 * FUNCTION NAME = vvUserBgndBankCopy
 *
 * DESCRIPTION
 *     On the 1st pass,
 *     Save returned client B bank number.
 *     Setup a VESA BIOS call to set the VRAM bank number to 0.
 *
 *     On all middle passes,
 *     Transfers the VRAM bank to virtual storage,
 *     Setup a VESA BIOS call to access the next VRAM bank.
 *
 *     On the last pass,
 *     Transfers the last VRAM bank to virtual storage,
 *     Setup a BIOS call to set VGA mode via vvUserBgndVGAModeSet
 *
 *****/
vvUserBgndBankCopy()
{
    /* New art */
    if( copy bank < 0 )
        save returned client B bank number
    else
        /* Prior art */
        transfer one VRAM bank to saved area
    /* New art */
    if( mode uses Linear Frame Buffer )
        transfer whole linear buffer to save area
        return to
            vvUserBgndVGAModeSet
    else
        increment copy bank number
        if( copy bank number < total banks )
            setup a VESA call to set copy A bank number.
            return to
                vvUserBgndBankCopySetBBank,
        else
            call vvUserBgndVGAModeSet directly
}

```

910

912

914

11/17

*FIG. 9D*

```

/*****
 *
 * FUNCTION NAME = vvUserBgndBankCopySetBBank
 *
 * DESCRIPTION
 *   Setup VESA BIOS call to set the copy B Bank Window,
 *   if it is needed for read/write operations.
 *
 *****/
vvUserBgndBankCopySetBBank()
{
    /* New art */
    Setup VESA BIOS call to set the copy B Bank Window,
    return to
    vvUserBgndBankCopy
}
/*****
 *
 * FUNCTION NAME = vvUserBgndVGAModeSet
 *
 * DESCRIPTION
 *   Setup a VGA BIOS call to set a VGA standard video mode (mode 12).
 *   This allows next operating system component manipulating the
 *   video hardware to assume the SVGA is a simple/standard VGA.
 *
 *****/
vvUserBgndVGAModeSet()
{
    /* New art */
    setup a VGA BIOS call to set a VGA standard video mode.
    return to vvUserBgndFinish
}
/*****
 *
 * FUNCTION NAME = vvUserBgndFinish
 *
 * DESCRIPTION
 *   Finish background switch in VDM's context
 *   Freeze VDM when in unemulatable (SVGA) video mode.
 *   Leave emulatable (VGA) video mode unfrozen.
 *
 *****/
vvUserBgndFinish()
{
    /* New art */
    restore client CPU register state
    /* Prior art */
    switch trapping behavior to emulation of hardware access
    freeze VDM when in unemulatable (SVGA) video mode.
}

```

916

918

920

FIG. 10

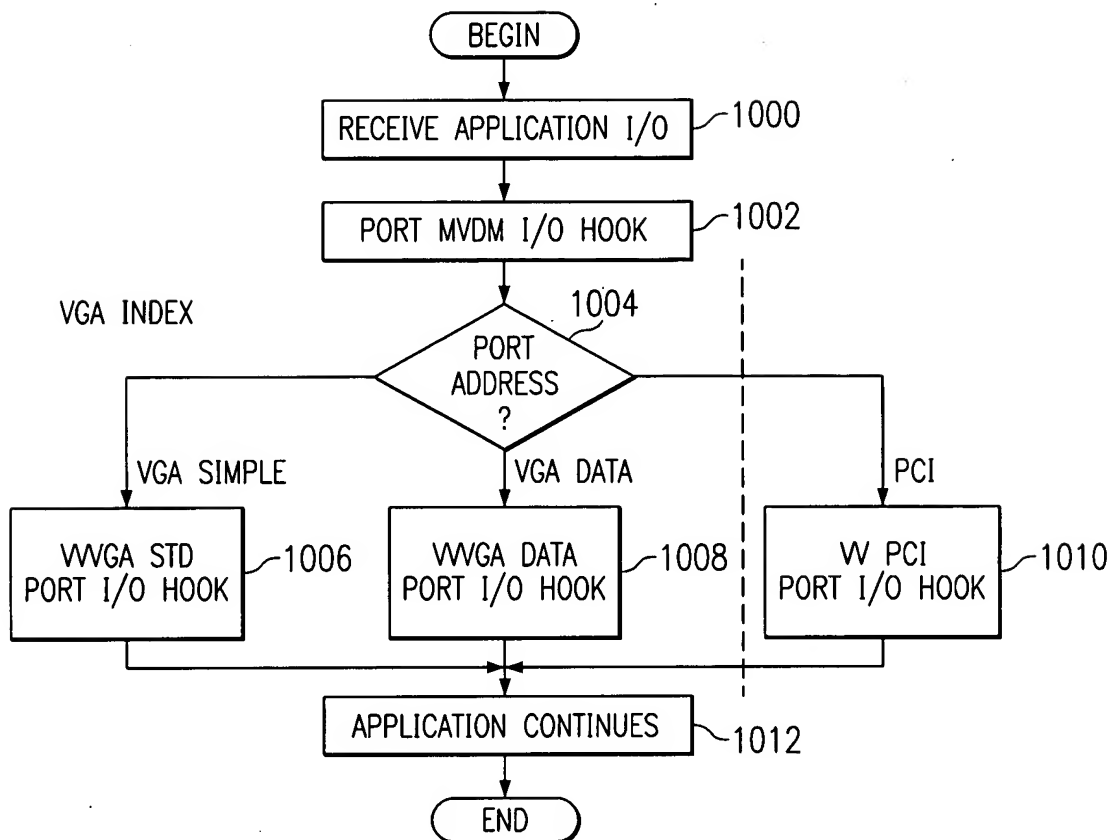


FIG. 11A

13/17

```

/*****
 *
 * FUNCTION NAME = wInit
 *
 * DESCRIPTION
 *   Initialization for virtual video driver
 *   called by mvdm at start of each VDM
 *   Most VESA BIOSes now provide PCI BIOS information too
 *
 *****/
wInit()
{
    /* Prior art: */
    register standard VGA I/O port address handlers with mvdm.
    /* New art: */
    make PCI BIOS call to get list of PCI BIOS I/O port addresses.
    for each PCI BIOS I/O port address
        register PCI BIOS I/O port address handler with mvdm.
}

/*****
 *
 * FUNCTION NAME = mvdmIOHook
 *
 * DESCRIPTION
 *   All client I/O instructions generate a hardware trap which comes here
 *   Handlers are generally all registered at the start of the VDM.
 *   Video port hooking is enabled in the background,
 *   and disabled in the foreground.
 *   Non-video hardware follows other algorithms based on the
 *   device driver requirements and sophistication.
 *
 *****/
mvdmPortIOHook()
{
    /* All prior art */
    if( registered handler for I/O port address
        && hooking enabled for I/O port address )
        call registered handler for I/O port address
    else
        do I/O directly.
}

/*****
 *
 * FUNCTION NAME = wVGASStandardPortIOHook
 *
 * DESCRIPTION
 *   Typical registered hook handler for VGA Standard I/O port address
 *   May be more complicated if I/O port not connected to a simple register
 *   Such as pair of I/O ports for an index and data register array
 *   Each I/O port address may have its own unique and differently
 *   coded handler to handle unusually behaving ports.
 *
 *****/
wVGASStandardPortIOHook()
{
    /* All prior art */
    if( input )
        return ( emulation state variable value for I/O port address )
    /* This goes into the client CPU register set */
    else /* output */
        Save output from client CPU register set
        into emulation state variable for I/O port address
        /* Will be used later to restore adapter contents */
        Adjust any other emulation state variables required by changes to this port
}

```

1100

1102

1104

FIG. 11B

```

/*****
*
* FUNCTION NAME = wVGADDataPortIOHook
*
* DESCRIPTION
*   Typical registered hook handler for VGA Standard I/O port address
*   as a part of index and data port handler pair.
*   Index port handler is usually a wVGASStandardPortIOHook.
*
*****/
wVGADDataPortIOHook()
{
    /* All prior art */
    if( input )
        return ( emulation state variable [index port state variable]
                value for I/O port address )
    /* This goes into the client CPU register set */
    else /* output */
        Save output from client CPU register set
        into emulation state variable [index port state variable]
        for I/O port address
        /* Will be used later to restore adapter contents */
        Adjust any other emulation state variables required by changes to this port
}
/*****
*
* FUNCTION NAME = wPCIPortIOHook
*
* DESCRIPTION
*   Registered by the virtual video device driver for a list
*   of port addresses provided by the PCI BIOS.
*   ONLY registered hook handler type for PCI BIOS I/O port address.
*   This represents a simple best guess to how a typical port works.
*   But it often does not absolutely correct emulation.
*   However it almost always suffices for emulating VGA modes.
*   This is NOT true of SVGA modes,
*   and this is why we freeze when in VESA modes in the background
*   so that the video adapter is not incorrectly emulated.
*   Emulation state variables used here
*   will NOT be used later to restore adapter contents,
*   because we do not know how port really works!
*   Instead we rely on the VESA BIOS calls to restore important registers.
*
*****/
wPCIPortIOHook()
{
    /* New art */
    if( input )
        return ( emulation state variable value for I/O port address )
    /* This goes into the client CPU register set */
    else /* output */
        save output from client CPU register set
        into emulation state variable for I/O port address
}

```

1106

1108

FIG. 12A

15/17

```

/*****
 *
 * FUNCTION NAME = wInt10PreHook
 *
 * DESCRIPTION
 *     Quick Return if not Mode Set,
 *     else transfer control to
 *
 *****/
wInt10PreHook()
{
    if( AH( pcrf ) == 0x00) /* VGA Mode Set */
    {
        /* Prior art */
        save client registers as last setmode registers
        wInt10Chain
    }
    else if( AX( pcrf ) == 0x4F02 ) /* VESA Mode Set */
    {
        /* Prior art: */
        save client registers as last setmode registers
        /* From here begins new art: */
        save VESA setmode number
        push client registers
        inject VESA call to get VESA BIOS SVGA INFO.
        return to wInt10VesaVbeInfoReturn
    }
    else
    {
        /* Prior art */
        wInt10Chain
    }
}

/*****
 *
 * FUNCTION NAME = wInt10VesaVbeInfoReturn
 *
 * DESCRIPTION
 *     Sets up for a VESA Mode query.
 *
 *****/
wInt10VesaVbeInfoReturn()
{
    save VESA BIOS SVGA INFO including total VRAM size.
    inject VESA BIOS call to get MODE INFO for new mode.
    return to
        wInt10VesaModeInfoReturn
}

/*****
 *
 * FUNCTION NAME = wInt10VesaModeInfoReturn
 *
 * DESCRIPTION
 *     Gets the VESA mode information from the Mode information block and
 *     copies it to the VDM's VESA mode information structure, and then
 *     sets up to do the actual VESA BIOS mode set to the VESA mode.
 *
 *****/
VOID HOOKENTRY wInt10VesaModeInfoReturn()
{
    pop client registers
    save current mode info as old mode info
    save VESA BIOS MODE INFO as current mode info
    ( includes mode dimension info )
    inject original setmode call to original VESA BIOS INT 10 handler
    return to
        wInt10VesaEndReturn
}

```

1200

1202

1204

```

/*****
 *
 * FUNCTION NAME = vInt10VesaEndReturn
 *
 * DESCRIPTION
 *     Does the post cleanup after the VESA BIOS mode set.
 *
 *****/
vInt10VesaEndReturn()
{
    if( AX( pcrf ) != VESA_FUNCTION_SUCCESS )
        restore current mode info from old mode info
    else if( background )
        freeze VDM
    vInt10Continue
}

/*****
 *
 * FUNCTION NAME = vInt10Chain
 *
 * DESCRIPTION
 *     Continue with client INT 10
 *
 *****/
vInt10Chain()
{
    call original (VGA/VESA) BIOS INT 10 handler
    return to vInt10Continue
}

/*****
 *
 * FUNCTION NAME = vInt10Continue
 *
 * DESCRIPTION
 *     return to client program
 *
 *****/
vInt10Continue()
{
    return to client program
}

```

FIG. 12B



```

/*****
 *
 * FUNCTION NAME = vvDsvModeUpdate
 *
 * DESCRIPTION
 *   Determine current mode dimensions
 *   These dimensions are used to determine:
 *   A) How much VRAM to save and restore for emulation switching
 *   B) How to draw current VRAM contents as a picture in a desktop window
 *
 *****/
vvDsvModeUpdate()
{
    if ( VESA MODE )
        /* New art: */
        calculate mode dimensions from info returned by previous VESA calls
        (current mode info)
    else
        /* Prior art */
        calculate mode dimensions from standard VGA registers
}

```

1214

FIG. 12C